OPENCARAC USER MANUAL
VERSION 1.0.1

Jonathan Certes

August 24, 2014

openCarac : Automatize your Spice simulator runnings
Copyright (C) 2014 Jonathan Certes
jonathan.certes@online.fr
http://opencarac.sourceforge.net

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see http://www.gnu.org/licenses/. Documents produced by *openCarac* are derivative works derived from the input used in their production; they are not affected by this license.

- TCL/TK is distributed under the terms of the BSD license.
- Doxygen is distributed under the terms of the GNU General Public License.
- Octave is distributed under the terms of the GNU General Public License.
- LaTeX is distributed under the terms of the LaTeX project public license.
- Ngspice is distributed under the terms of the modified BSD license.
- SMASH is a registered trademark of Dolphin Integration.
- Linux is distributed under the terms of the GNU General Public License and other licenses.
- Windows is a registered trademark of Microsoft.

This program is a work in progress. Some to-dos are listed in the Doxygen^{TM} documentation. More is surely needed. You are invited to report bugs, missing items, wrongly described items, bad English style etc.

What is new since version 1.0?

Thanks to users feedback and intense use of *openCarac*, many problems have been detected and corrected since the previous release. Here is a description of what has been implemented.
The developer would like to say a special thank to Julien Gilleron for his feedback and bugs reports.

Bugs fixing:

- Avoids a crash when printing an error message if *.MODELLIN* or *.MODELWIN* is not set correctly.
- Avoids a crash when .SIMU directive is not properly defined.
- Avoids a crash when a simulation or a carac cannot be launched because of previous errors.
- Avoids a crash during results extraction when a result must be checked and a value is *NaN*.
- Avoids a crash when creating a full check summary when two results have the same name and only one has to be checked.
- Avoids a crash when a simulation must run whilst setting a parameter that does not exist.
- Avoids a crash when the line that follows the ***MODEL*** marker does not contain 3 elements.
- Avoids printing erroneous values after SMASH^{TM} *.mes files parsing.
- Uncomment *.CHECKOP* and *.CHECKMEAS* directives in the *carac.conf* file of Ngspice^{TM} buffer example.
- *carac.conf* file parser now ignores empty lines properly.
- Move the Hook for custom execution between simulation running and results files parsing to wait for the end of simulations execution.
- When creating an archive, *model* file and *libparam* file are now copied properly in the archive.
- Octave script functions are now corrects even if no step has to be described.
- In LaTeX destination files, use escape syntax for both underscore and backslash.
- Colorize the min and max values when a *checkmeas* is set in LaTeX summary lines just like in HTML summary ones.

Improvements:

- Improving SMASH^{TM} *.mes file parser to detect if the step comes from an ”alter”. If it does, adding the number of the alter after a point ”.” in the step value.
- Check TCL version before running, do not run if version is lower than 8.5.
- Keep the results formated like the simulator gave them.
- Execution time improvement for results extraction and archive generation.
- Accepts parentheses in *checkmeas* names.
- When verifying if a carac can be launched, make sure that the main file is located next to the carac configuration file and not in a sub-directory.

Changing behavior:

- When running simulations the classic way (not by step), they run all before executing the hooks. This makes openCarac creating more folders at the same time but also give a better yield when simulations can run on various computers. If disk space is an issue, running by steps is still a better solution.
- In archives, a different Octave file is created for each simulation name with different parameters. Octave matrices now contain the results from every steps, also including ”oneStepAndValues”. *checkmeas* matrices are now correct for every step. No more Octave summary file is created.

Contents

1 Introduction **4**
- 1.1 Brief description . . . 4
- 1.2 openCarac output files . . . 4
- 1.3 FAQ . . . 5

2 Your first carac **6**
- 2.1 Add openCarac to your environment . . . 6
- 2.2 Organize your spice circuit files . . . 8
- 2.3 Run a carac . . . 10

3 openCarac usage **11**
- 3.1 openCarac execution details . . . 11
  - 3.1.1 Generation of an archive . . . 11
  - 3.1.2 Generation of *checkmeas* full summary files . . . 12
- 3.2 carac.conf file syntax . . . 12
  - 3.2.1 .CHANGE directive . . . 14
  - 3.2.2 .CHECKMEAS directive . . . 14
  - 3.2.3 .CHECKOP directive . . . 15
  - 3.2.4 .CORNER directive . . . 15
  - 3.2.5 .EXTRACTOP directive . . . 16
  - 3.2.6 .LIBPARAMLIN directive . . . 17
  - 3.2.7 .LIBPARAMWIN directive . . . 17
  - 3.2.8 .LINLIBPARAM directive . . . 18
  - 3.2.9 .LINMODEL directive . . . 18
  - 3.2.10 .MODELLIN directive . . . 18
  - 3.2.11 .MODELWIN directive . . . 19
  - 3.2.12 .NETLIST directive . . . 19
  - 3.2.13 .PARAM directive . . . 20
  - 3.2.14 .PARAMETERS directive . . . 20
  - 3.2.15 .RESET directive . . . 20
  - 3.2.16 .SIMU directive . . . 21
  - 3.2.17 .SIMULATIONS directive . . . 22
  - 3.2.18 .SIMULATOR directive . . . 22
  - 3.2.19 .WINLIBPARAM directive . . . 22
  - 3.2.20 .WINMODEL directive . . . 22
- 3.3 Execution options . . . 23

4 Expert mode **25**
- 4.1 Your own simulator commands . . . 25
  - 4.1.1 Organization of the *customProcedures.tcl* file . . . 25
  - 4.1.2 *execute simulator* procedure . . . 26
  - 4.1.3 *custom hook* procedure . . . 26

5 Known problems **27**
- 5.1 Troubles using openCarac with Ngspice^{TM} on Windows^{TM} . . . 27

Chapter 1

Introduction

Abstract: Welcome to openCarac user manual. This document's purpose is to introduce the tool that is openCarac and explain how to use it in order to get the best of it to automatize your simulations with your Spice simulator. We assume that the reader already has basic knowledges about circuit simulation using Spice. In this chapter is given a brief description of this tool and what it gives to the user, then frequently asked question are also answered.

1.1 Brief description

In electronics, the designer who creates a circuits has to verify its functionality by doing various simulations, such as distortion measurement, noise calculation, stability evaluation, and so on; and if he wants to test his circuit's robustness, also has to vary his devices corners (typical, slow, fast) and his simulation conditions by tuning parameters such as temperature, power supply voltage or bias current. When crossing all those conditions, it results an huge amount of simulations to run that can be difficult to handle; also, Spice simulators generate formated results files that are not always easy to treat or summarize and getting the information out them can be difficult to automatize.

This is when openCarac could be useful; this tool has been implemented to automatize this task by running the user's favorite Spice simulator in predefined conditions. The user only has to create a "carac configuration file" in a simulation environment organized to this purpose and run openCarac in order to make all his simulations run and have his data summarized in a results folder.

Indeed, by using openCarac, the designer will be in possession of easily exploitable files to treat his data and create quality document out of them as described in section 1.2.

Note: Since many Spice simulator exist and do not accept the same syntax, do not give the results files in the same format, a choice had to be made to make openCarac compatible with one of them. In consequence, openCarac is not compatible with every Spice simulators. Indeed, its predilection Spice simulator is Ngspice^{TM}, but it has been implemented to be also compatible with Dolphin Integration SMASH^{TM}.

1.2 openCarac output files

When simulating using a Spice simulator, the designer usually extracts data values out of his work using a *.MEAS* or *.MEASURE* directive. This command permits the user to get an exact value of what he wants to measure. This is the kind of data openCarac focuses on: this tool is used to save and organize what is given by the simulator, not to calculate anything by checking the plot curves. Indeed, after having executed the simulator, openCarac parses the results files and creates its own output results files by summarizing what it has found.
In those results files, the user can find the following formats:

- Tables in HTML format: navigate easily through your simulations results using hyper-links in a web browser.
- LaTeX code files: make quality documents with you results saved in tables.
- GNU Octave^{TM} script files: treat easily your data by calculating anything you want with your simulation results saved into matrices.

1.3 FAQ

Where does the name ”openCarac” come from?

Since openCarac is a tool to simulate an electronic circuit in various conditions, it is used to characterize it; so, the developer of this tool being French has chosen this name for the following reason: ”carac” comes from the French word *caractérisation*, meaning ”characterization”.
Of course, the prefix ”open” has been added since it is an open-source project.

Why is Ngspice^{TM} openCarac’s predilection simulator?

We have seen that openCarac is an open-source project, as its purpose is to be an extension to a Spice simulator, it was logical that the first one to be chosen was also part of an open-source project. Besides, since openCarac is coded using TCL and that an API (application programming interface) called TCLspice also coded in TCL is available with Ngspice^{TM}, it becomes possible to easily merge the two approaches for the user to customize the environment.
These two characteristics make Ngspice^{TM} a first-class choice for openCarac to be compatible with.

Can openCarac be compatible with other simulators?

There are many Spice simulators available and being compatible with all of them represents a huge amount of work: they all accept a different syntax for their directives, the output files formats are always different. So, adapting openCarac to many of them is the goal that has been fixed but, this project being a work-in-progress, we will have to wait a little to see it compatible with other simulators.

Why is there a duck printed in my results files?

This is a frequently asked question which still remains unanswered. If one day you appear to have an answer, please let us know.

Chapter 2

Your first carac

Abstract: In order to use openCarac, the user has to adapt his environment to make it compatible with. Since this installation requires a little comprehension of how does the tool work, this chapter describes the steps to prepare your environment to welcome openCarac whilst explaining the reasons to do so; get ready to benefits of its advantages.

2.1 Add openCarac to your environment

First, know that openCarac is a cross-platform[1] TCL script which can be added to your environment to automatize simulations runnings by executing a Spice simulator command with different conditions of a circuit. Then, it must be executed through a TCL interpreter and a compatible Spice simulator must be available.

So, before being able to run openCarac, make sure that a TCL interpreter is available; openCarac works fine with version 8.5; and so is a Spice simulator present in the following list:

- Ngspice^{TM} 26
- Dolphin Integration SMASH^{TM} 6.2.0

Run openCarac:

Basically, as any TCL script, openCarac must be executed through a TCL interpreter, then you must know the command to execute it. We suggest that you run your interpreter from your command line interface, because when openCarac terminates, "exit" tcl command is executed and TCL interpreter quits; if it has been executed from a command line interface, only the interpreter does and previously printed text remains visible.

So, execute the command to run your TCL interpreter followed by the path to access openCarac script. This may look like one of the following commands.

If you are using Linux:

```
$ tclsh ~/openCarac/tcl/openCarac.tcl
```

If you are using Windows:

```
> C:\tcl\bin\tclsh.exe C:\openCarac\tcl\openCarac.tcl
```

If you have an error message, make sure that both TCL interpreter command and openCarac script path are correct.

Otherwise, you can see the copyrights printed in your command line interface, this means that the execution has occurred perfectly, even if you see an error message such as *"Error, no configuration file found. No carac running."*: this only means that you have not configured your simulator environment, this will be done in section 2.2.

[1] openCarac has been tested under both Linux and Windows NT.

Define your simulation environment:

Congratulations, your are now able to run openCarac. But you may wonder how does it know how to run your simulator? The answer is simple: it does not. So, openCarac is not yet integrated to your environment: you must then tell it what is the command to execute your favorite simulator.

To do so, give an argument to your TCL interpreter to make openCarac create a configuration file, adapt and execute the following command[2].

If you are using Linux:

```
$ tclsh ~/openCarac/tcl/openCarac.tcl --configure
```

If you are using Windows:

```
> C:\tcl\bin\tclsh.exe C:\openCarac\tcl\openCarac.tcl --configure
```

As a consequence, openCarac will create a folder named "*.openCarac*" and a configuration file into your home directory[3] which you can modify to adapt it to your environment. A message is printed and gives the location of this file.

Note that another file "*customProcedures.tcl*" has also been generated, its usage is for experts with TCL language and is described in chapter 4.

In this configuration file are described the settings of openCarac: parameters are defined and set to particular value, comments are also present starting with a dash (#). Now, just open this configuration file with your favorite text editor, this may look like something as follows:

```
# Default output log file:
logFile = stdout

# Default simulator:
simulator = ngspice

# Simulators commands:
ngspiceCommand = ngspice
smashCommand   = smash

# Default preferences:
runBySteps  = no
archive     = no
fullsummary = no
filesCopy   = yes
custom      = no
```

These are the definitions of the parameters to set openCarac environment: on each line, the name of the parameter is located between the beginning of the line and the equal (=), its value is then defined between the same equal and the end of the line. You can modify them in order to change the commands to execute the simulators.

Here is a brief description of each one of the parameters:

- logFile : When executing openCarac, informations are printed, you can choose to print them in a log file or in the standard output. This parameter allows the user to define the path to a log file to print the informations in. This path can be absolute or relative to the location of execution. If its value is "*stdout*", informations are printed in the standard output. Note that this parameter defines the default value, this can be altered when executing openCarac.
- simulator : Since openCarac is compatible with various simulators, you must specify the one you use; this is the selection of the default simulator. This also can be altered when executing openCarac.

[2] In case you are running openCarac through a TCL interpreter different from Tclsh, you can set your argument by setting the variable "*argv*" to "$--configure$". See section 3.3 for more informations.

[3] Under windows NT, it is created into your user directory into the "*Documents and settings*" folder.

- ngspiceCommand : This parameter is used to defined the command to execute simulator Ngspice^{TM}, we suggest that you use an alias or an absolute path to run the command. Since aliases do not exist under Windows, if using it you do not have a choice.
- smashCommand : Just like the previous one, this parameter is used to define the command to execute the simulator; this one is to run Dolphin Integration SMASH^{TM}.
- runBySteps : If value is set to *yes*, it changes *openCarac* default behavior: when running openCarac, caracs will be executed by steps. This has the same effect as selecting option *runbysteps* when executing *openCarac*, see section 3.3 for more details.
- archive : If value is set to *yes*, it changes *openCarac* default behavior: when running openCarac, it activates the generation of archives after results extraction. This has the same effect as selecting option *archive* when executing *openCarac*, see section 3.3 for more details.
- fullsummary : If value is set to *yes*, it changes *openCarac* default behavior: when running openCarac, it activates the generation of chekmeas summary files after results extraction. This has the same effect as selecting option *fullsummary* when executing *openCarac*, see section 3.3 for more details.
- filesCopy : If value is set to *no*, it changes *openCarac* default behavior: after having every simulation run, openCarac does not copy output simulator files into the *openCaracFiles* directory next to the *carac.conf* configuration file. This has the same effect as selecting option *nofilescopy* when executing *openCarac*, see section 3.3 for more details.
- custom : If value is set to *yes*, it changes *openCarac* default behavior: execute the simulations using custom procedures defined in the *customProcedures.tcl* file located in your home directory. This has the same effect as selecting option *custom* when executing *openCarac*, see section 3.3 for more details.

So, in order to make openCarac compatible with your simulation environment, just change the values of the commands to execute the simulators to be the ones you run in your environment. Here is an example of a configuration file to use openCarac under Windows:

```
# Default output log file:
logFile = ..\myOpenCaracLogFile.txt

# Default simulator:
simulator = ngspice

# Simulators commands:
ngspiceCommand = C:\spice\bin\ngspice.exe
smashCommand   = C:\Dolphin\smash\bin\smash.exe

# Default preferences:
runBySteps = no
archive    = no
filesCopy  = yes
custom     = no
```

As soon as this configuration file has been properly filled, it seems that openCarac has been added to your environment, you are now ready to use it!

2.2 Organize your spice circuit files

The purpose of openCarac is to automatize simulations runnings using your simulator: when characterizing a circuit, the designer must run various type of simulations such as open loop AC type, transient, noise... Also, he must tune simulation parameters: bias current, power supply voltage, temperature... He may make his components corners vary too: slow, fast, best, worst, whatever the definition is.

So, openCarac provides a way to run every needed simulations in the previously designed conditions. In order to do so, the user must organize his spice files to make them fully compatible with openCarac; a way to organize them has been chosen and is described as follows:

- A main file contains inclusions of other files or loadings of libraries.
- Spice devices models are defined in only one file containing various libraries[4]; in each one of them are the definitions of the devices depending on the corner.
- Simulation conditions are established into one only file containing various libraries, each one of them are used to sweep parameters[5] in the desired conditions. Having various libraries allows to sweep more or less parameters, depending on the simulation conditions.
- Each simulation is defined into a separated file that must be included into the control file only once at the time. This permits to separate every simulation. The extension of such a file must be ".inc".

In the previously defined control file, loadings of libraries for both models and parameters, so as inclusions of simulation files, can be operated by openCarac at the location stipulated by markers. So the user finally has to add markers into his control file and then, openCarac will use them to tune it in various simulations. Since comment character in Spice syntax is the star (*), each marker will be composed of a keyword surrounded by couples of stars; in those keywords, we can find:

- ****MODEL****: marker to explicit the location where to change the libraries to tune the models.
- ****PARAM****: marker to explicit the location where to change the libraries to modify the simulation conditions.
- ****SIMU****: marker to explicit the location to include the simulation file with extension ".inc".

Every time that one of these keywords is encountered by openCarac, the following line in the file is altered and a simulation is executed.

Then, after having prepared the files like as previously described, the user will be able to configure openCarac to make it change the library to load for the models selection, the parameters to change the simulation conditions or the file to include. Here is an example of a Spice control file that allows to run a carac using openCarac:

```
** openCarac example circuit **
*******************************

**MODEL**
.LIB '../../foundryModels/models.lib' TYPICAL

**PARAM**
.LIB '../../parameters/conditions.lib' TYP

**SIMU**
.include 'op.inc'
*.include 'openLoop.inc'
*.include 'transient.inc'
*.include 'noise.inc'

** The netlist is a separated file:
.include 'nampli.nsx'

.END
```

In this example is represented the control file that contains the three keywords. We can note that the loaded models are located in a library called "*TYPICAL*" in a file named "*models.lib*". Respectively, simulation conditions are defined in a library called "*TYP*" in a file named "*conditions.lib*". In here, four kinds of simulations are needed to characterize the circuit, they are separated into four different files: *op.inc*, *openLoop.inc*, *transient.inc* and *noise.inc*.

So, when carac is going to be executed, models library name "*TYPICAL*" might be altered, so does parameters library "*TYP*" and four different simulations can be run.

[4] You can define or access Spice libraries using the *.LIB* directive.

[5] Parameters sweep permits to run the same simulation more than once whilst varying the so called parameter.

2.3 Run a carac

Now that your Spice circuit files are properly organized, your first carac is now ready to run: you are now able to automatize the execution of various simulations when altering both models and parameters libraries to load.

Note that openCarac will not include every simulation files located in the same folder, only you can choose the list of these simulations. So, in order to chose the names of the libraries to load for both models and parameters and the simulations to run, the last step is to add a *carac configuration file* in the same folder as your Spice control file; such a file must be named "*carac.conf*" in order to be recognized by openCarac. Its syntax is described in chapter 3.

Supposing that we want to run a carac with the previous Spice control file where we want to alter both models and parameters libraries and run all of the simulations; let's pretend that models libraries names are as follows:

- TYPICAL
- WORST
- BEST

and parameters libraries are:

- TYP
- VDD_IBIAS_TEMP (where supply voltage, bias current and temperature vary by steps)

Then the "*carac.conf*" configuration file may look like as follows (details about the syntax are given in chapter 3):

```
## openCarac example configuration file

.CHANGE myCarac

.MODEL   TYPICAL WORST BEST
.PARAM   TYP VDD_IBIAS_TEMP
.SIMU    op openLoop transient noise
```

To run your carac through your command line interface, get to the folder where your *carac.conf* file is located and run openCarac according to your command as described in section 2.1; all the informations about the running are printed in the standard output or the log file you have specified, openCarac will execute your simulator, extract the results and then exit.

When executing, openCarac will run the four simulations *op*, *openLoop*, *transient* and *noise* by including the files and alter models and parameters libraries. So, since there are in this case 4 simulations, 3 models libraries and 2 parameters libraries, there will be $4 \times 3 \times 2 = 24$ simulations running.

As soon as every simulation is terminated, openCarac parses your simulator results files looking for the values of measures (data values extracted using a *.MEAS* or *.MEASURE* directive) and creates HTML files, LaTeX code and GNU Octave scrips. These files are saved into a *openCaracFiles* folder created next to your *carac.conf* file.

If you can see these results files, it seems that you have perfectly integrated openCarac into your environment and that you are now able to automatize your simulations runnings. Congratulations!
Detailed informations about openCarac runnings and carac configurations are given in chapter 3.

Chapter 3

openCarac usage

Abstract: In this chapter are described all the details about openCarac and the ways to benefit the whole potential of this tool by presenting the possible options to configure it.

3.1 openCarac execution details

First, in order to know how to customize your carac runnings, openCarac execution is to be described: this section explains what happens when a carac is executed so you can tune your environment in order get the best of it.

Since it is about running many times your favorite simulator by altering libraries loadings and files inclusions using Spice directives, the links to access these files must be preserved; and, since simulations executions may take some time, it can be useful to keep access to your files when you have chosen to run a carac. So, choice has been made to create temporary folders to run your simulations into when openCarac is executed: a copy of necessary files is created in directories generated for this purpose next to your simulation folder. In this case, relative links to access Spice libraries and included files remain corrects[1]. Note that if you want to execute a lot of simulations, every temporary folders are created first and then the simulator is executed into each one of them. These directory are not created one at the time in order to keep possible the capability to execute the simulations simultaneously with different computers, a solution to do so is described in chapter 4.

When creating these temporary folders, openCarac modifies your Spice files to alter libraries loadings and files inclusions. Note that you can also alter parameters values when choosing your simulations as described in subsection 3.2.16. It uses the markers that are available in your main file to chose the line to alter (see section 2.2), only lines following these keywords are modified. So, beware your Spice inclusion directives locations before running your carac and make sure that they are always following a keyword if they are to be altered; otherwise, more than one file inclusion could appear and your simulations may not work.

Your simulator command which has been defined in your openCarac system configuration file (see section 2.1) is executed by sourcing the copy of your main file into every temporary folder and outputs are redirected into files. When every simulation has run, these files are being parsed to get the values of your measures[2] and saved into a *openCaracFiles* directory into your original simulation folder; then temporary directories are deleted. Gathered informations are used to create results files such as HTML or LaTeX tables and GNU Octave^{TM} scripts, also located in the *openCaracFiles* directory.

3.1.1 Generation of an archive

When running openCarac using the *archive* option (see section 3.3 for more informations about execution options), archive folders will be created after results extraction, containing every files openCarac needs to run; meaning every file copied in temporary folders. Also, if defined using *.LIBPARAM** and *.MODEL** directives, the library files to load are also saved into the archive folder. Two folders with the same names as directories containing these library files are created and the

[1] Of course, this also work with absolute paths.

[2] A measure is what is get by your simulator through a *.MEAS* or *.MEASURE* directive.

files are copied into them. The name of these archive folders are formated using keyword *archive*, the date and time when they are created and the name of library files and their parent directories.

Also, openCarac results files are created in it: these files are re-formated in order to contain every informations and results from various caracs. Note that in an archive folder are joined every carac created with the same configuration file that have the same *.LIBPARAM** and *.MODEL** library files loaded.

3.1.2 Generation of *checkmeas* full summary files

When running openCarac using the *fullsummary* option (see section 3.3 for more informations about execution options), full summary files will be generated after results extraction. In such a file can be found a tabular summarizing every results having a *checkmeas* in carac configuration files. Every result having the same name will have its value printed in the same line of the tabular; columns are separated from their files configuration: every carac having the same file containing the parameters libraries to load and the same file containing the models libraries to load are joined together to make the creation of a new column.

These summary files are created in the current directory, meaning where *openCarac* has been executed. The names of these files are formated using keyword *openCarac_Check_Summary* and the date and time when they are created.

3.2 carac.conf file syntax

To configure your carac running, you have at your disposition what is called a *carac configuration file* that must be named *carac.conf*, this file allows to configure your carac by setting the alterations of models and parameters libraries or simulations files inclusions. But it also has other characteristics that permit to:

- Run more than one carac of your circuit
- Change the path to access your libraries files (for models or simulation conditions)
- Alter your netlist by including an other file.
- Extract devices parameters from your operating point.
- Add verifications on your measures values.
- Add verifications on some voltages value at operating point.

Global arrangement:

Here is a description of *carac.conf* files syntax:
Any comment can be added by starting with a dash (#), any text located between a dash and the end of the line is going to be ignored. *carac.conf* files syntax is not case sensitive. The characteristics of your caracs can be tuned using directives starting with a point (.) and a keyword just like with Spice syntax. Any line can be continued on the next one using character plus (+).

A carac is defined by the values of all directives located between two *.CHANGE* directives (see the description of this directive for more explanations), or between the beginning of the file and the first *.CHANGE* directive (in this case, your carac name will be "*myCarac*", but it does not exist if no simulation has been stipulated), or between the last *.CHANGE* directive and the end of the file.

When having various caracs defined in the same file, values of directives set in previous ones remain set to the same value unless they are overloaded. You can overload the value with an empty string by setting the directive alone with no value if you want reset it.

Here is an example a *carac.conf* file:

```
# This is a comment

.CHANGE TheFirstCarac

.PARAM TYP

.SIMU    op
+        openLoop    # This is an other comment.
+        transient

.CHANGE TheSecondCarac

.PARAM SWEEP

# Value of .SIMU directive remains the same.
```

In this example, two caracs will be executed: in the first one, library "*TYP*" of the simulation conditions file will be loaded; in the second one, library "*SWEEP*" will be loaded; in both of them, three simulations are going to be executed by including respectively files *op.inc*, *openLoop.inc* and *transient.inc*.

3.2.1 .CHANGE directive

Description:

This directive is used to define the name of a carac and to delimit its description. Unless it is the first one encountered in a *carac.conf* file when no simulation has been stipulated, the previous defined carac is set when this directive appears. When using this directive, the user choose the name of his carac using the parameter `<name>` and then can change all its configuration using every available directives. The carac definition ends when an other *.CHANGE* directive is encountered or when End Of File is reached. So, a carac configuration is usually defined between two *.CHANGE* directives and the name of the carac is the one set in the first of these two.

If a parameter has been set in a previous carac configuration and the same directive is not re-used in the current carac definition, the previous value is also used for this carac. Indeed this directive does not reset the previous configurations and allows only to define the parameters the user wants to alter: any directive which is re-used permits to overload the value of the previous one.

If the user wants to reset the value of a carac parameter, he may use the same directive used as before with no value at all.

Syntax:

```
.CHANGE <name>
```

Parameters:

`<name>` String: defines the name of the carac which is configured after the *.CHANGE* directive.

3.2.2 .CHECKMEAS directive

Description:

This directive is used to verify that the measures in the simulations are in accordance to some specifications: it permits to define the name of the measure to check and the minimum and maximum tolerated values. So, when extracting the results, openCarac will check the conformity of the results and prints them with a format in accordance. For instance, results will be colorized in green or red into HTML tables or check matrices will be available in GNU Octave^{TM} scripts.

To use this directive, the user must just add a triplet of parameters to define respectively the name of the measure, its minimum and maximum tolerated values. More than one triplet can be defined if they are more than one measure to check.

Syntax:

```
.CHECKMEAS <name> <min> <max> [<name> <min> <max>]*
```

Parameters:

`<name>` String: defines the name of the measure the user wants to check the value of.
`<min>` Real: minimum tolerated value for the measure to check.
`<max>` Real: maximum tolerated value for the measure to check.

Example:

In the following example is shown the configuration of a carac where both simulations *op.inc* and *openLoop.inc* are used and two measures named "*phi_margin*" and "*gain_dc*" are defined in file *openLoop.inc*.
It permits to verify that the measured value of "*phi_margin*" is higher than 1.0472 and lower than 3.1416 and that measured value of "*gain_dc*" is higher than 60 and lower than 100.

```
.CHANGE myCarac

.SIMU    op
+        openLoop    # AC analysis where measures to check are defined.

.CHECKMEAS  phi_margin 1.0472 3.1416   # 60/180*pi = 1.0472
+           gain_dc    60     100
```

3.2.3 .CHECKOP directive

Description:

Since openCarac is used to sum up the values of measures and measuring a voltage at operating point is not available with every simulator, directive *.CHECKOP* can be defined. When set, this directive makes openCarac parse the simulator results files looking for voltage values at operating point. Just like when using directive *.CHECKMEAS* (see subsection 3.2.2), it permits to define the name of the voltage to check the value of and the minimum and maximum tolerated values. Conformity of the results is checked and they are printed with a format in accordance.

To use this directive, the user must add a triplet of parameters to define respectively the name of the net to get the voltage value of, its minimum and maximum tolerated values. More than one triplet can be defined if they are more than one voltage value to check.

Syntax:

```
.CHECKOP <net> <min> <max> [<net> <min> <max>]*
```

Parameters:

- `<net>` String: defines the name of the net the user wants to check the voltage value of.
- `<min>` Real: minimum tolerated value for the voltage value of the net to check.
- `<max>` Real: maximum tolerated value for the voltage value of the net to check.

Example:

In the following example is shown the configuration of a carac where a simulation *op.inc* is used and two nets named "*IN1*" and "*IN2*" are defined in the netlist file.
It permits to extract the voltage values of nets "*IN1*" and "*IN2*" at operating point and verify that the value of the voltage on net "*IN1*" is higher than 2.25 and lower than 2.75 and that measured value of the voltage on net "*IN2*" is higher than 0 and lower than 5.

```
.CHANGE myCarac

.SIMU    op

.CHECKOP  IN1 2.25 2.75
+         IN2 0    5
```

3.2.4 .CORNER directive

Description:

When running a carac where models libraries are to be altered, a list of library names to load has to be specified (see section 2.2), these library names are called *corners*. In order to define the list of corners to load, directive *.CORNER* has to be used. A list of strings, libraries names, separated by spaces is to be added to define the corners to load.

Syntax:

```
.CORNER <name> [<name>]*
```

Parameters:

`<name>` String: defines the name of the library to load.

Example:

In the following example is shown the configuration of a carac where a simulation *op.inc* is used and is simulated in three corners.

```
.CHANGE myCarac

.SIMU    op
.CORNER  TYP FAST SLOW
```

3.2.5 .EXTRACTOP directive

Description:

Since openCarac is used to sum up the values of measures and extracting a device parameter at operating point is not available with every simulator, directive *.EXTRACTOP* can be defined. When set, this directive makes openCarac parse the simulator results files looking for devices parameters values at operating point. It permits to define the name of the simulation the operating point must be get from and the name of the parameters to extract.

If nothing else is done, the given parameter is going to be extracted from every devices available in the simulator output files and data is printed in the results file just like it was a measure extraction. In order to filter a list of devices to extract the parameters from, the user can add a file "*devices.list*" in the same directory as the "*carac.conf*" file where a list of devices is defined; if this file is present, parameters will be extracted only from the devices having the same names as given in this file.

Syntax:

```
.EXTRACTOP <simulation> <parameter> [<simulation> <parameter>]*
```

Parameters:

`<simulation>` String: name of the simulation to extract the parameter from.
`<parameter>` String: this defines the name of the parameter to extract from the devices.

Note: the simulation name must also be present in the list given to the *.SIMU* directive as defined in subsection 3.2.16.

Example:

In the following example is shown the configuration of a carac where three simulations *op.inc*, *openLoop.inc* and *tran.inc* are used, parameter *VDS* will be extracted from devices in results files when simulating with file *op.inc* and parameter *ID* will be extracted from devices in results files when simulating with file *openLoop.inc*.

```
.CHANGE myCarac

.SIMU      op       openLoop tran
.EXTRACTOP op       VDS
+          openLoop ID
```

If a filter must be applied on two devices, the following "*devices.list*" file can be added in the same directory:

```
M.XAMP.MDP2
M.XAMP.MDP1
```

3.2.6 .LIBPARAMLIN directive

Description:

Since openCarac allows to alter the loading of a library for simulation conditions parameters, the user must load a file containing all the possible libraries. If various library files are available to define different conditions for a circuit, directive *.LIBPARAMLIN* permits to define a Linux^{TM} format path to access an other file containing parameters libraries. In this case, it becomes possible to change the file containing the parameters libraries to load.

This directive can be completed by the usage of *.LIBPARAMWIN* which allows to define a Windows^{TM} format path to access the same file (see subsection 3.2.7); in this case, the user can define both paths to access the same file, each of them compatible with respectively Linux^{TM} and Windows^{TM}.

An alias to this directive is also available: *.LINLIBPARAM* directive, see subsection 3.2.8.

Syntax:

```
.LIBPARAMLIN <path>
```

Parameters:

`<path>` String: Linux^{TM} format path to access the file containing the parameters libraries to load.

Example:

In the following example is shown the configuration of a carac where the file containing the parameters libraries to load is altered. This works under both Linux^{TM} and Windows^{TM}.

```
.CHANGE myCarac

.LIBPARAMLIN  /media/device/folder/conditions.lib
.LIBPARAMWIN  E:\folder\conditions.lib
```

3.2.7 .LIBPARAMWIN directive

Description:

Since openCarac allows to alter the loading of a library for simulation conditions parameters, the user must load a file containing all the possible libraries. If various library files are available to define different conditions for a circuit, directive *.LIBPARAMWIN* permits to define a Windows^{TM} format path to access an other file containing parameters libraries. In this case, it becomes possible to change the file containing the parameters libraries to load.

This directive can be completed by the usage of *.LIBPARAMLIN* which allows to define a Linux^{TM} format path to access the same file (see subsection 3.2.6); in this case, the user can define both paths to access the same file, each of them compatible with respectively Linux^{TM} and Windows^{TM}.

An alias to this directive is also available: *.WINLIBPARAM* directive, see subsection 3.2.19.

Syntax:

```
.LIBPARAMWIN <path>
```

Parameters:

`<path>` String: Windows^{TM} format path to access the file containing the parameters libraries to load.

Example:

In the following example is shown the configuration of a carac where the file containing the parameters libraries to load is altered. This works under both Linux^{TM} and Windows^{TM}.

```
.CHANGE myCarac

.LIBPARAMWIN  E:\folder\conditions.lib
.LIBPARAMLIN  /media/device/folder/conditions.lib
```

3.2.8 .LINLIBPARAM directive

This directive is an alias to *.LIBPARAMLIN* directive. See *.LIBPARAMLIN* description in subsection 3.2.6 for more informations.

3.2.9 .LINMODEL directive

This directive is an alias to *.MODELLIN* directive. See *.MODELLIN* description in subsection 3.2.10 for more informations.

3.2.10 .MODELLIN directive

Description:

Since openCarac allows to alter the loading of a library for models corners, the user must load a file containing all the possible corners libraries. If various library files are available to define different models for the devices, directive *.MODELLIN* permits to define a Linux^{TM} format path to access an other file containing models corners libraries. In this case, it becomes possible to change the file containing the models libraries to load.

This directive can be completed by the usage of *.MODELWIN* which allows to define a Windows^{TM} format path to access the same file (see subsection 3.2.11); in this case, the user can define both paths to access the same file, each of them compatible with respectively Linux^{TM} and Windows^{TM}.

An alias to this directive is also available: *.LINMODEL* directive, see subsection 3.2.9.

Syntax:

```
.MODELLIN <path>
```

Parameters:

`<path>` String: Linux^{TM} format path to access the file containing the models libraries to load.

Example:

In the following example is shown the configuration of a carac where the file containing the models libraries to load is altered. This works under both Linux^{TM} and Windows^{TM}.

```
.CHANGE myCarac

.MODELLIN  /media/device/folder/models.lib
.MODELWIN  E:\folder\models.lib
```

3.2.11 .MODELWIN directive

Description:

Since openCarac allows to alter the loading of a library for models corners, the user must load a file containing all the possible corners libraries. If various library files are available to define different models for the devices, directive *.MODELWIN* permits to define a Windows^{TM} format path to access an other file containing models corners libraries. In this case, it becomes possible to change the file containing the models libraries to load.

This directive can be completed by the usage of *.MODELLIN* which allows to define a Linux^{TM} format path to access the same file (see subsection 3.2.10); in this case, the user can define both paths to access the same file, each of them compatible with respectively Linux^{TM} and Windows^{TM}.

An alias to this directive is also available: *.WINMODEL* directive, see subsection 3.2.20.

Syntax:

```
.MODELWIN <path>
```

Parameters:

`<path>` String: Windows^{TM} format path to access the file containing the models libraries to load.

Example:

In the following example is shown the configuration of a carac where the file containing the models libraries to load is altered. This works under both Linux^{TM} and Windows^{TM}.

```
.CHANGE myCarac

.MODELWIN  E:\folder\models.lib
.MODELLIN  /media/device/folder/models.lib
```

3.2.12 .NETLIST directive

Description:

Sometimes, it can be useful to run the same simulations with different netlists and duplicate all the files cannot be the best way to do so. A solution to re-use the simulations files is offered by openCarac: if the user gets the part of the circuits he wants to alter in a separated file with extension *.nsx*, it can be changed when running the carac as soon as an inclusion of the *.nsx* file is done in the main control file.

To do so, *.NETLIST* directive permits to define a list of *.nsx* file names which will be loaded during the carac. Note that only the first occurrence of *.nsx* file inclusion is replaced in the main file when running the carac.

Syntax:

```
.NETLIST <name> [<name>]*
```

Parameters:

`<name>` String: defines the name of the *.nsx* file to include.

Example:

In the following example is shown the configuration of a carac where both file *nampli.nsx* and *pampli.nsx* will be alternatively included in the main control file.

```
.CHANGE myCarac

.NETLIST nampli pampli
```

3.2.13 .PARAM directive

Description:

When running a carac where simulation conditions parameters libraries are to be altered, a list of library names to load has to be specified (see section 2.2), these library names permits to define your simulation conditions. In order to define the list of parameters libraries to load, directive *.PARAM* has to be used. A list of strings, libraries names, separated by spaces is to be added to define the conditions parameters libraries to load.

Syntax:

```
.PARAM <name> [<name>]*
```

Parameters:

`<name>` String: defines the name of the library to load.

Example:

In the following example is shown the configuration of a carac where a simulation *op.inc* is used and is simulated with two conditions parameters libraries.

```
.CHANGE myCarac

.SIMU    op
.PARAM   TYP SWEEP
```

3.2.14 .PARAMETERS directive

This directive is an alias to *.PARAM* directive. See *.PARAM* description in subsection 3.2.13 for more informations.

3.2.15 .RESET directive

Description:

When configuring various caracs using *.CHANGE* directive as described in subsection 3.2.1, the user only needs to re-define what he wants to alter, the configurations of every directive remain identical if not set again. If a directive must be set to its default value, it is possible to set this directive with no parameter: having an empty string is interpreted as the directive has never been set.

Sometimes, a carac configuration may be so different as the previous one that many directives have to be reset; in such a case, it becomes possible to use *.RESET* directive. This makes every directive except *.CHANGE* placed before reset to its default value.

Parameters:

This directive is used with no parameter.

Syntax:

```
.RESET
```

Example:

In the following example is shown the configuration of four caracs where:

- In the first carac, a simulation *op.inc* is used where measure "myMeas" is checked.
- In the second carac, a simulation *transient.inc* is used where measure "myMeas" is also checked.
- In the third carac, *.CHECKMEAS* directive is reset and simulation *openLoop.inc* is used with no measure checked.
- In the fourth carac, nothing is done because *.RESET* directive also resets the *.SIMU* directive placed before.

```
.CHANGE firstCarac
.SIMU      op
.CHECKMEAS myMeas 0 1

.CHANGE secondCarac
.SIMU      transient

.CHANGE thirdCarac
.RESET
.SIMU      openLoop

.CHANGE fourthCarac
.SIMU      op
.RESET
```

3.2.16 .SIMU directive

Description:

When running a carac where various simulations have to be run, the user must separate the simulations definitions in various files with extension *.inc* (see section 2.2). So, to define the list of simulations to run, directive *.SIMU* has to be used and a list of strings, names of files with *.inc* extensions, separated by spaces has to be added to define the files where the simulations are configured to include.

Also, since it can be useful to modify the values of parameters, the user can make each simulation name being followed by couples of parameters name and value defined between parentheses. What is defined in a field delimited by parentheses will determine the parameters to alter when running the simulation once, various couples can be defined to make various parameters vary. If the user wants to run various times the same simulation when altering the parameters with other values or other parameters, others fields delimited by parameters can be defined.

Each time, the first string is considered as the name of the simulation and what is defined between the parentheses is considered as a configuration. If various configurations are described after a simulation name, the same simulation name is considered and the simulation will be run various times.

Syntax:

```
.SIMU <name> [(<paramName> <paramValue> [<paramName> <paramValue>]*)]*
+    [<name> [(<paramName> <paramValue> [<paramName> <paramValue>]*)]*]*
```

Parameters:

`<name>`	String: defines the name of the file with *.inc* extension to include.
`<paramName>`	String: defines the name of the parameter to alter.
`<paramValue>`	String: defines the value to give to the parameter.

Example:

In the following example is shown the configuration of a carac where four simulations *op*, *openLoop*, *transient* and *noise* are used by including respectively files *op.inc*, *openLoop.inc*, *transient.inc* and *noise.inc*.

Simulation *openLoop* will run twice, the first time when altering both parameters "millerValue" and "loadValue" to respectively $20f$ and $10k$; and the second time when altering parameter "millerValue" to $35f$ and by keeping originally defined value of parameter "loadValue". Other simulations will run once with no parameter alteration.

```
.CHANGE myCarac

.SIMU op
+     openLoop  (millerValue 20f loadValue 10k) (millerValue 35f)
+     transient noise
```

3.2.17 .SIMULATIONS directive

This directive is an alias to *.SIMU* directive. See *.SIMU* description in subsection 3.2.16 for more informations.

3.2.18 .SIMULATOR directive

Description:

In case a carac configuration can be run only with a specific simulator, directive *.SIMULATOR* can be used to force the selection of the simulator to run the carac with. Its effect is to overload the simulator selection that may have been done through openCarac default configuration or running options.

If the specified simulator is not supported by openCarac, an error message is printed and the directive remains ignored.

Syntax:

```
.SIMULATOR <name>
```

Parameters:

`<name>` String: defines the name of the simulator to select.

Example:

In the following example is shown the configuration of a carac where simulator *Ngspice* is selected and simulation *op* will run in three corners.

```
.CHANGE myCarac

.SIMULATOR NGSPICE
.SIMU      op
.CORNER    TYP FAST SLOW
```

3.2.19 .WINLIBPARAM directive

This directive is an alias to *.LIBPARAMWIN* directive. See *.LIBPARAMWIN* description in subsection 3.2.7 for more informations.

3.2.20 .WINMODEL directive

This directive is an alias to *.MODELWIN* directive. See *.MODELWIN* description in subsection 3.2.11 for more informations.

3.3 Execution options

When executing a TCL interpreter such as *tclsh*^{TM}, it is possible to add a vector of arguments during TCL execution which then can be used to alter openCarac's behavior. For instance, it has been shown in section 2.3 that openCarac must be executed in a folder containing a file named *carac.conf*; but, it is possible to give as an a argument the path to access such a configuration file as defined bellow:

If you are using Linux:

```
$ tclsh ~/openCarac/tcl/openCarac.tcl simulation/carac.conf
```

If you are using Windows:

```
> C:\tcl\bin\tclsh.exe C:\openCarac\tcl\openCarac.tcl simulation\carac.conf
```

These arguments are defined in a vector named *argv* in TCL. So, in case you are using a TCL interpreter which does not allow to pass arguments during the loading of openCarac script file, it is possible to set variable *argv* before sourcing the main file. This may look like as shown bellow:

```
% set argv "--ngspice [file join {simulation} {carac.conf}]" ; source openCarac.tcl
```

The variable *argv* is in fact a list of arguments separated by spaces: it is then possible to give various arguments to openCarac at the same time when you are executing it. If more than one path to a *carac.conf* file is given, all the files will be taken into consideration for simulation.

More than paths to *carac.conf* files, you can also add execution options to change the behavior of openCarac. An execution option is interpreted so when starting with either two minus signs "−−" and a keyword or one minus sign "−" and a letter. Note that almost every keyword has its equivalent as a letter, that arguments are case-sensitive and that the order you pass them has no matter for openCarac. Here is a list of every keyword available and their descriptions:

archive (-a): activates the generation of archives after results extraction, see subsection 3.1.1 for more informations.

check (-c): only check that caracs are going to work properly and do not run the simulations. This makes openCarac load only the bare necessary simulations include files while ignoring Spice simulations directives and only try opening the circuits; since simulation state is printed in openCarac results file, it permits to anticipate the functionality of a carac without loading every possible simulations.

configure (-C): makes openCarac create a configuration file defining your settings in your home directory. Also creates a *customProcedures.tcl* which you can modify to change openCarac simulations runnings behavior, see chapter 4 for more informations. The path to access the settings file is printed to the standard output and openCarac exits.

custom (-t): execute the simulations using custom procedures defined in the *customProcedures.tcl* file located in your home directory instead of openCarac default commands. See chapter 4 for more informations.

debug (-d): debug mode, do not remove the temporary folders after the simulations. As described in section 3.1, when running, openCarac creates temporary folders next to your simulation directory and removes them after the simulations runnings; this option disables the deletion of such temporary folders.

extractres (-e): do not run the simulations and only creates output results files from data available in the *openCaracFiles* directory next to the *carac.conf* configuration file. Note that data has to be available by having previously run a carac and that extracted results are function of the configuration detailed in the *carac.conf* configuration file. This permits to create results after having run a carac in various passes.

fullsummary (-f): activates the generation of *checkmeas* full summary files after results extraction, see subsection 3.1.2 for more informations.

help (-h): prints a quick help in the standard output and then exits.

logfile $< arg >$**:** when running openCarac, useful informations are printed either in the standard output (default configuration) or in a file defined by your openCarac environment settings, see the description of openCarac settings file in section 2.1 for more informations. This execution option allows to define the path to a log file to print the informations in instead of the one selected in the openCarac settings file. Argument $< arg >$ represents the path to this log file; it can be absolute or relative to the location of execution.

multicarac $< arg >$**:** instead of passing a list of paths to *carac.conf* files as a vector of arguments to add various caracs to the running list, it is possible do print all these paths into a text file and use *multicarac* option. When using this option, this file is read by openCarac and all the *carac.conf* files having their paths in it are added to the running list. Argument $< arg >$ represents the path to the file containing the locations of every *carac.conf* file; it can be absolute or relative to the location of execution.

ngspice (-n): selects Ngspice^{TM} simulator instead of the one selected by your openCarac environment settings, see the description of openCarac settings file in section 2.1 for more informations.

nofilescopy (-N): after having every simulation run, openCarac copies output simulator files into the *openCaracFiles* directory next to the *carac.conf* configuration file; if this option is selected, this copy does not happen. This might be useful to save disk space in case the simulator files are heavy.

runbysteps (-S): when various caracs have to run, openCarac default behavior is that every needed temporary folder is created (as described in section 3.1) before starting any simulation and, once every simulation has terminated, execute results extractions. In case simulation times are too high and results file are needed quick, it is possible to make openCarac repeat the operation for every carac one at the time: this behavior is selected when using option *runbysteps*. In this case, when executing one carac, temporary folders are created, simulations are executed, temporary folders are removed and results are extracted; and so on with other caracs.

smash (-s): selects Dolphin Integration SMASH^{TM} simulator instead of the one selected by your openCarac environment settings, see the description of openCarac settings file in section 2.1 for more informations.

Chapter 4

Expert mode

Abstract: openCarac aims to be a tool to simplify the runnings of simulations and the treatment of results; it uses TCL (Tool Command Language) to treat files or data and integrates the execution of Spice simulators. If in your environment you want to use other functionality such as option provided by your software's API (Application Programming Interface) or Shell/DOS scripts for any other automation, openCarac provides custom procedures to execute your simulator. In this chapter is explained how to use these custom procedures, knowledges about TCL language are required to use them.

4.1 Your own simulator commands

Custom procedures can be selected instead of openCarac default procedures when executing using option "$--custom$" as described in section 3.3; this makes openCarac use the procedures defined in a file named *customProcedures.tcl* available in the *.openCarac* folder in your home directory[1].

So, in order to create your own procedures to execute your simulator, you can edit this file to change how openCarac makes your simulator running.

4.1.1 Organization of the *customProcedures.tcl* file

In this file are available a *namespace* and two *procedures* by simulator. Note that, to keep openCarac working properly, you must not alter the names of these *namespace* and *procedures*. When executing openCarac in the custom mode, these procedures are executed function of the selected simulator, here is a description of their use:

- execute simulator procedure: After having created temporary folders to execute the simulations into, simulator is executed by openCarac using the absolute path to the main file located in this directory. This procedure allows to customize the execution of the simulator.
- custom hook procedure: In case the simulator is executed in background, it may make openCarac remove the temporary folder and extract the results before the end of the simulations. To avoid this kind of inconvenient, this custom hook procedure can be defined to wait for an event: nothing will happen until this procedure does end. Note that when running, openCarac starts by executing every simulations of your carac and then executes this hook once.
- customExecution namespace: This namespace can be used to save informations about the simulation running through the *execute simulator* procedure and to access these informations into the *custom hook* procedure. It remains unused by openCarac itself.

To understand how these procedures must be used, here a simplified code describing the algorithm of simulations runnings in openCarac using Ngspice^{TM}:

[1] If this folder is not available, just run command "*openCarac --configure*" as defined in section 2.1

```
foreach theCarac $theCaracList {

  foreach theSimulation [carac::getSimulationsList $theCarac] {
    set simulationMainFilePath [simulation::getSimulationmainfile $theSimulation]
    customExecution::executeNgspiceSimulator $simulationMainFilePath $logFile
  }

}

foreach theCarac $theCaracList {

  customExecution::ngspiceHook $logFile

  foreach theSimulation [carac::getSimulationsList $theCarac] {
    set simulationFolder [simulation::getTemporaryfolderpath $theSimulation]
    file delete -force $simulationFolder
  }

}
```

4.1.2 *execute simulator* procedure

When executing this procedure, openCarac gives the absolute path to the main file to source, it does not change the location of execution: this command is executed in the folder you have executed openCarac. As a consequence, every file created during the execution of this procedure must have its path absolute or, if relative, it must be relative to the location of execution.

Also, to keep openCarac's behavior correct, beware that it parses the simulator output files and that file names and extensions must be respected: output results file must be named with the same name as the Spice main file with extension *.log* and what is printed in the standard output must be saved in a file with the same name and extension *.rpt*. Note that this behavior can be different with other simulator than Ngspice^{TM} in case you cannot choose the output files names and extensions.

When executed, this procedure can receive some parameters depending of what the simulator needs. It can be used to modify your simulator running with the behavior of your choice:

- **mainFilePath**: String: Absolute path to the main file that must be loaded with the simulator.
- **simulationType**: String: word defining the type of simulation, this type has been identified by parsing the Spice include files.
- **isMonteCarlo**: Boolean: defines if this is a Monte-Carlo simulation; if this simulation has not been set to be a Monte-Carlo, given value is zero. It has been identified by parsing the Spice include files.
- **isCheckModeOn**: Boolean: defines if openCarac is running in check mode, this has been set by adding parameter $--check$ while executing openCarac, see section 3.3 for more informations.
- **logFile**: Log file buffer to print informations in openCarac's log using "puts" command.

4.1.3 *custom hook* procedure

After having executed *execute simulator* procedure for each simulation of the carac, this *custom hook* procedure is executed for each available carac. You can use the namespace to access any informations saved during *execute simulator* to check the state of your simulations in case they are running in background and make this procedure wait for every simulations to have terminated.

In case your simulations are running in foreground, openCarac will not continue running until every simulation has been completed, this procedure becomes useless and may do nothing.

Note that you can choose to return anything with this procedure, it remains unused by openCarac.

Chapter 5

Known problems

Abstract: openCarac being a work in progress, you may be experiencing some problems when using it in particular conditions. This chapter describes known problems that have not been corrected yet or that cannot be corrected because of compatibility reasons and workarounds to avoid them.

5.1 Troubles using openCarac with Ngspice^{TM} on Windows^{TM}

In case your simulation environment is on *Windows* using *Ngspice*, you may see pop-up windows appear when running openCarac; also, your *Ngspice* report files may remain empty.
This problem is caused by the difference of command line interface (CLI) between openCarac and *Ngspice*; indeed, if you run openCarac through your *DOS* CLI and call *Ngspice* with its graphical user interface (GUI), messages are not printed in the same location. So, when openCarac reads what *Ngspice* has returned to your CLI to print it into the report file, nothing is found because the informations have been printed into *Ngspice* GUI.

In order to make openCarac work properly on *Windows*, you must stipulate *Ngspice* not to use its GUI and execute it from your operating system CLI: during *Ngspice* configuration, before its compilation, you must not specify the `--with-windows` option.

As a consequence, this will deactivate the GUI of *Ngspice* and you will have to run your *DOS* command line interface to run *Ngspice* into it. Informations will then be printed into the standard output and openCarac will be able to catch them and write a correct report.